CrocoMarine Training Final Project

Task 2: Distance Estimation Using YOLOv8

Made by: Fares Hazem Mohamed Shalaby

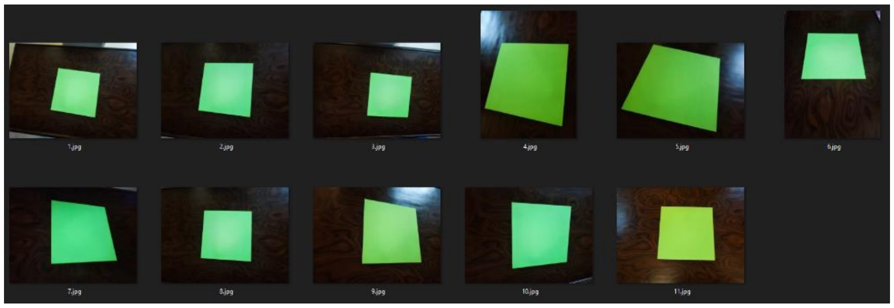
Table Of Contents

Introduction

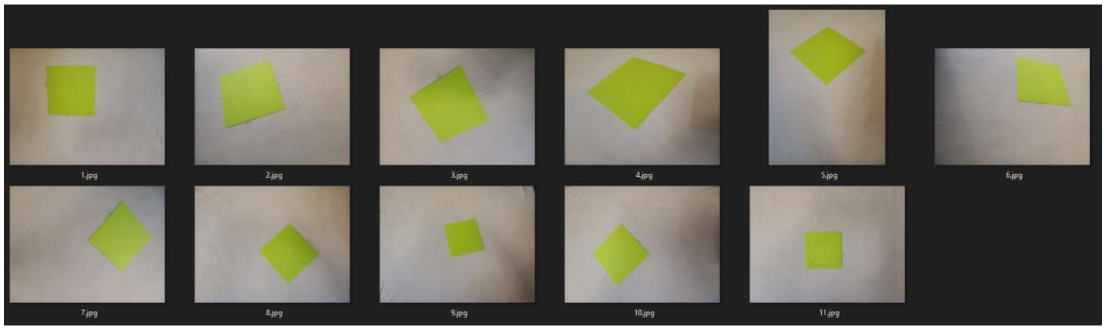
In this project, my objective is to implement a distance estimation system between a real-time camera and a designated 15x15 square, referred to as a "box," detected by the YOLOv8 model. To achieve this, I plan to train the YOLOv8 model using a custom dataset that I will capture and prepare with the assistance of the Roboflow website. The trained model will then be integrated into a Python code, enabling its application on a real-time camera feed. The distance estimation will be computed using a formula, Further details about this formula will be explained in the following sections. This project aims to showcase the practical application of object detection and distance estimation in a real-world scenario.

<u>Data Collection</u>

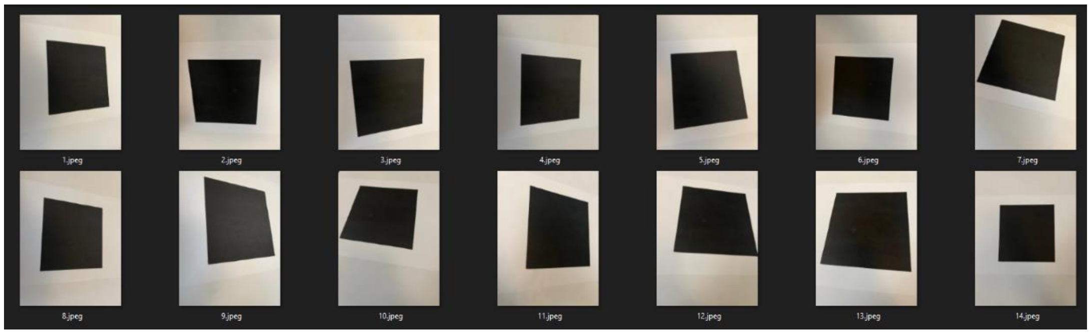
For the data collection phase, I conducted multiple captures to enhance the quality of the dataset. Initially, I collected data featuring green squares against a wooden background. However, these instances posed challenges as the YOLOv8 model struggled to detect the squares effectively due to noise in the background.



In response, I collected a subsequent dataset with green squares against a white cloth background. Although this dataset facilitated smoother detection in Roboflow, it presented issues during YOLOv8 model training. Challenges arose from variations in lighting conditions and the presence of wrinkles in the cloth, impacting overall performance.

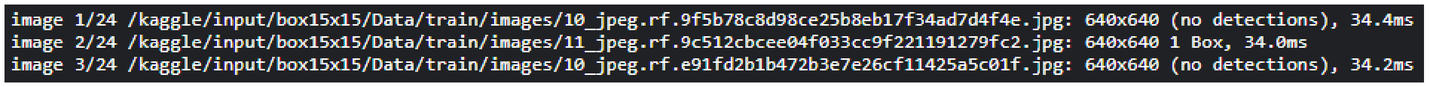


The final dataset involved capturing images of a black square against a white paper background. Although initial YOLOv8 model performance was suboptimal, I experimented with various adjustments. Initially attempting Photoshop corrections for potential color depth and lighting issues, the model's performance deteriorated. However, subsequent modifications, which will be detailed later, led to a significant improvement in performance compared to the previous datasets.



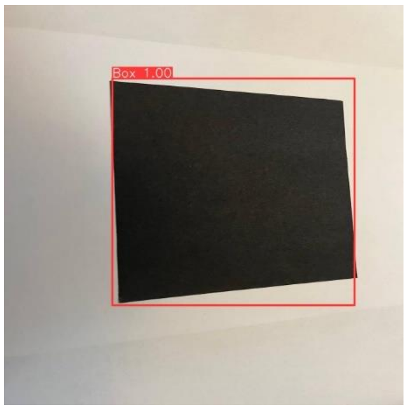
YOLOv8 Model Training

For the YOLOv8 model training, my initial attempt involved utilizing a model commonly employed by the Kaggle community. However, its performance on the final dataset was less than satisfactory. To enhance the model's capabilities, I explored data augmentation and preprocessing techniques. Additionally, I experimented with modifying the number of epochs and batches, eventually achieving satisfactory results with 50 epochs and 8 batches. The model successfully detected two out of three images in the test set.

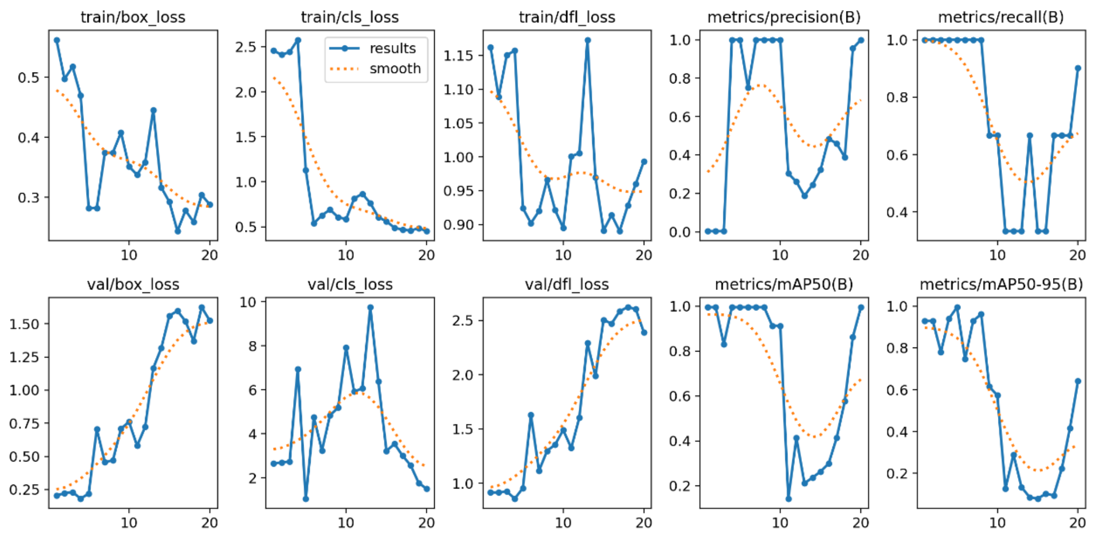


```
image 1/24 /kaggle/input/box15x15/Data/train/images/10_jpeg.rf.9f5b78c8d98ce25b8eb17f34ad7d4f4e.jpg: 640x640 (no detections), 34.4ms
image 2/24 /kaggle/input/box15x15/Data/train/images/11_jpeg.rf.9c512cbcee04f033cc9f221191279fc2.jpg: 640x640 1 Box, 34.0ms
image 3/24 /kaggle/input/box15x15/Data/train/images/10_jpeg.rf.e91fd2b1b472b3e7e26cf11425a5c01f.jpg: 640x640 (no detections), 34.2ms
```

Seeking further optimization, I explored additional resources on YouTube and discovered Nicolai Nielsen's alternative approach to implementing the YOLOv8 model using Google Colab. Following his methodology, I witnessed improved performance, with the model successfully detecting all test data after just 20 iterations. Notably, this success was achieved using the YOLOv8m model.



At the end, I displayed some results to understand the model more.



Applying YOLOv8 Model on Real-time Camera

After successfully training the YOLOv8 model, I saved and downloaded it from Google Colab. The next step involved applying the trained model to a real-time camera feed, enabling predictions on objects captured by the webcam.

During this process, a challenge surfaced where the model exhibited a tendency to identify numerous objects as squares, potentially causing inaccuracies in predictions. To address this, I increased the confidence level threshold to 0.999. This adjustment significantly improved the model's performance, ensuring more accurate and reliable predictions in real-time camera applications.

Distance Estimation Algorithm Implementation

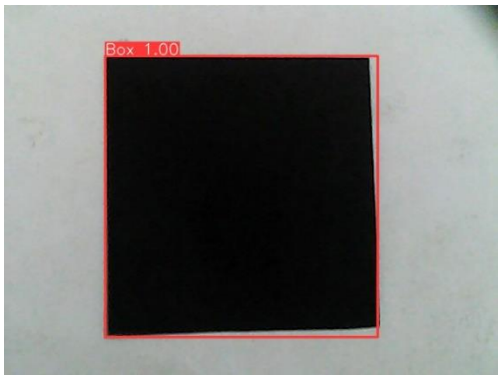
To implement the distance estimation algorithm, I utilized the formula:

$$distance = \frac{size\_of\_black\_square\_in\_real\_world \times focal\_length}{size\_of\_black\_square\_in\_real\_world}$$

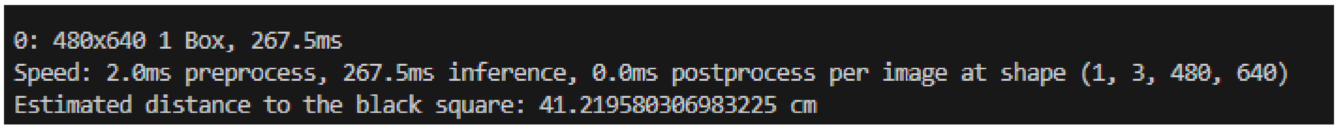
where:

- $focal\_length = 1140$
- $\text{size\_of\_black\_square\_in\_real\_world} = 15.0$

Given that the size of the black square is known to be 15x15 units, I initially attempted to calculate the remaining variables using formulas. While the calculation for size_of_black_square__in_image yielded accurate results, determining the correct value for focal_length proved challenging. As a result, I manually adjusted the focal length until reaching an optimal value of 1140, ensuring accurate distance estimations.



To validate the algorithm's effectiveness, I conducted initial tests on static images before applying it to real-time camera feeds. This iterative approach allowed for fine-tuning and optimization, ultimately resulting in reliable distance estimations in dynamic environments.



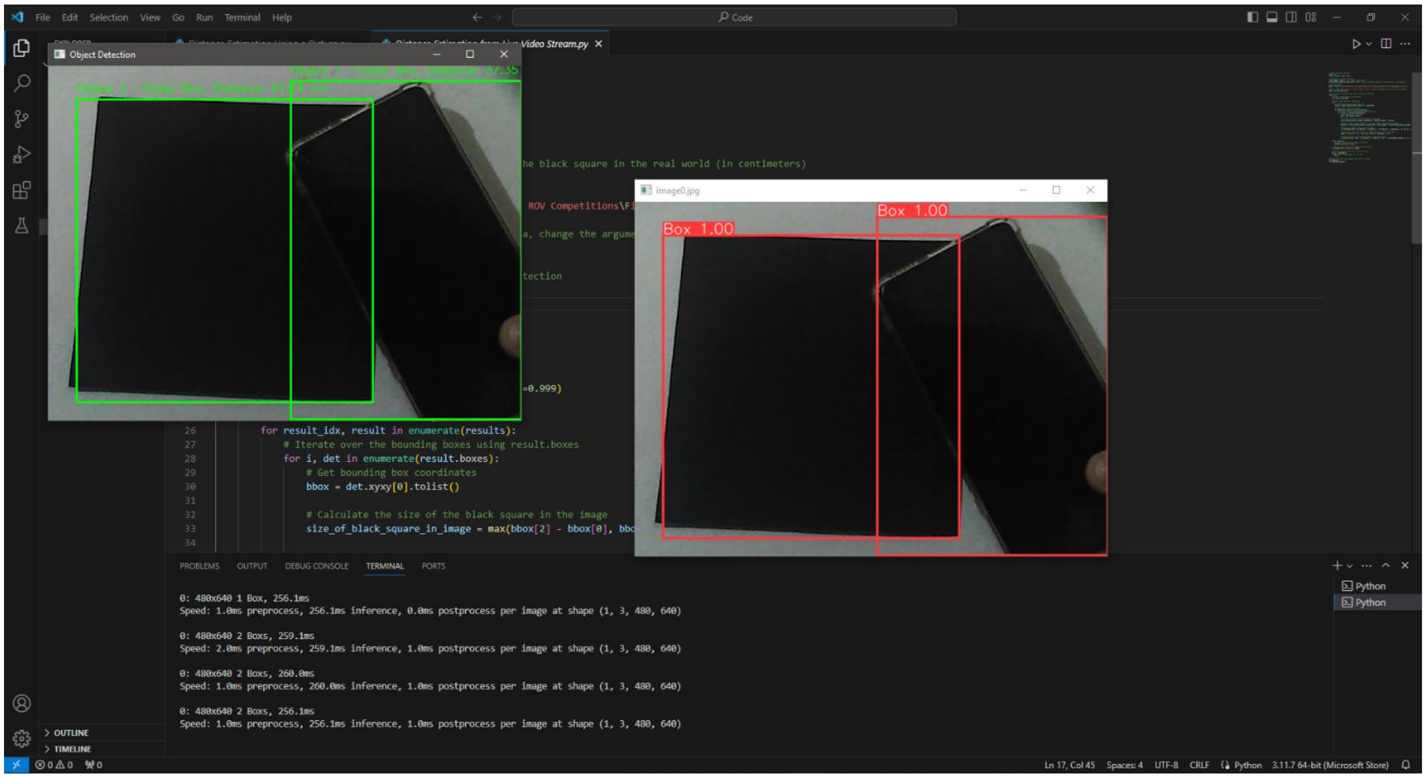
```
0: 480x640 1 Box, 267.5ms
Speed: 2.0ms preprocess, 267.5ms inference, 0.0ms postprocess per image at shape (1, 3, 480, 640)
Estimated distance to the black square: 41.219580306983225 cm
```

Integration of All Components

In the Integration of All Components, the YOLOv8 algorithm and the distance estimation formula were combined with real-time camera functionality. However, several challenges arose during this integration. Initially, the program would crash if no detection occurred, prompting the implementation of error handling using try and catch mechanisms.

Another issue surfaced where the program inaccurately assumed the presence of only one box at a time. This was addressed by iterating over the result boxes generated by the model, accommodating scenarios with multiple detected objects.

To enhance user experience, the distance estimation was incorporated into the real-time preview window, providing comprehensive feedback. These adjustments collectively contribute to a more robust and user-friendly integration of all components.


File Edit Selection View Go Run Terminal Help
Code
Video Stream.py
Object Detection
image0.jpg
Box 1.00
Box 1.00
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
Python
Ln 17, Col 45 Spaces: 4 UTF-8 CRLF Python 3.11.7 64-bit (Microsoft Store)

```python
he black square in the real world (in centimeters)
 ROV Competitions\F
a, change the argume
tection
=0.999)
        for result_idx, result in enumerate(results):
            # Iterate over the bounding boxes using result.boxes
            for i, det in enumerate(result.boxes):
                # Get bounding box coordinates
                bbox = det.xyxy[0].tolist()

                # Calculate the size of the black square in the image
                size_of_black_square_in_image = max(bbox[2] - bbox[0], bbc
```

```
0: 480x640 1 Box, 256.1ms
Speed: 1.0ms preprocess, 256.1ms inference, 0.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 2 Boxs, 259.1ms
Speed: 2.0ms preprocess, 259.1ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 2 Boxs, 260.0ms
Speed: 1.0ms preprocess, 260.0ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 2 Boxs, 256.1ms
Speed: 1.0ms preprocess, 256.1ms inference, 1.0ms postprocess per image at shape (1, 3, 480, 640)
```